1.Max and min

Program:

a={9,9,7,5,5,0,9}

b=max(a)

c=min(a)

print("max",b)

print("min",c)

2.merge sord

Program:

def mergeSort(a):

print("Splitting ",a)

if len(a)>1:

mid = len(a)//2

lefthalf = a[:mid]

righthalf = a[mid:]

#recursion

mergeSort(lefthalf)

mergeSort(righthalf)

i=0

j=0

k=0

while i < len(lefthalf) and j < len(righthalf):

if lefthalf[i] < righthalf[j]:

a[k]=lefthalf[i]

i=i+1

else:

a[k]=righthalf[j]

j=j+1

k=k+1

while i < len(lefthalf):

a[k]=lefthalf[i]

i=i+1

k=k+1

while j < len(righthalf):

a[k]=righthalf[j]

j=j+1

k=k+1

a = [54,26,93,17,77,31,44,55,20]

mergeSort(a)

print(a)

3. floyds algorithm

Program:

def floyd\_warshall(dist\_matrix):

n = len(dist\_matrix)

# Initialize shortest\_path matrix with distances

shortest\_path = [[dist\_matrix[i][j] for j in range(n)] for i in range(n)]

# Floyd-Warshall algorithm

for k in range(n):

for i in range(n):

for j in range(n):

if shortest\_path[i][j] > shortest\_path[i][k] + shortest\_path[k][j]:

shortest\_path[i][j] = shortest\_path[i][k] + shortest\_path[k][j]

return shortest\_path

def print\_shortest\_paths(dist\_matrix, shortest\_path):

n = len(dist\_matrix)

for i in range(n):

for j in range(n):

if shortest\_path[i][j] == float('inf'):

print(f"Shortest path from city {i} to city {j}: No path")

else:

print(f"Shortest path from city {i} to city {j}: Distance = {shortest\_path[i][j]}")

# Example usage:

if \_\_name\_\_ == "\_\_main\_\_":

# Example distance matrix representing cities and their distances

dist\_matrix = [

[0, 5, float('inf'), 10],

[float('inf'), 0, 3, float('inf')],

[float('inf'), float('inf'), 0, 1],

[float('inf'), float('inf'), float('inf'), 0]

]

print("Distance Matrix before Floyd's Algorithm:")

for row in dist\_matrix:

print(row)

print()

shortest\_path = floyd\_warshall(dist\_matrix)

print("Distance Matrix after Floyd's Algorithm (Shortest paths between all pairs of cities):")

for row in shortest\_path:

print(row)

print()

print("Shortest Paths:")

print\_shortest\_paths(dist\_matrix, shortest\_path)